*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the [Jupyter Notebook FAQ (https://www.coursera.org/learn/python-machine-learning/resources/bANLa)](https://www.coursera.org/learn/python-machine-learning/resources/bANLa) course resource.*

Assignment 1 - Introduction to Machine Learning

For this assignment, you will be using the Breast Cancer Wisconsin (Diagnostic) Database to create a classifier that can help diagnose patients. First, read through the description of the dataset (below).

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()

# print(cancer.DESCR)
```

The object returned by `load_breast_cancer()` is a scikit-learn Bunch object, which is similar to a dictionary.

In [2]:
```
cancer.keys()
```

Out[2]:
```
dict_keys(['DESCR', 'target_names', 'feature_names', 'target', 'data'])
```

Question 0 (Example)

How many features does the breast cancer dataset have?

This function should return an integer.

In [3]:
```
# You should write your whole answer within the function provided. The autograder will call
# this function and compare the return value against the correct solution value
def answer_zero():
    # This function returns the number of features of the breast cancer dataset, which is an integer.
    # The assignment question description will tell you the general format the autograder is expecting
    return len(cancer['feature_names'])

# You can examine what your function returns by calling it in the cell. If you have questions
# about the assignment formats, check out the discussion forums for any FAQs
answer_zero()
```

Out[3]: 30

Question 1

Scikit-learn works with lists, numpy arrays, scipy-sparse matrices, and pandas DataFrames, so converting the dataset to a DataFrame is not necessary for training this model. Using a DataFrame does however help make many things easier such as munging data, so let's practice creating a classifier with a pandas DataFrame.

Convert the sklearn.dataset `cancer` to a DataFrame.

This function should return a `(569, 31)` DataFrame with

columns =

```
['mean radius', 'mean texture', 'mean perimeter', 'mean area',
'mean smoothness', 'mean compactness', 'mean concavity',
'mean concave points', 'mean symmetry', 'mean fractal dimension',
'radius error', 'texture error', 'perimeter error', 'area error',
'smoothness error', 'compactness error', 'concavity error',
'concave points error', 'symmetry error', 'fractal dimension error',
'worst radius', 'worst texture', 'worst perimeter', 'worst area',
'worst smoothness', 'worst compactness', 'worst concavity',
'worst concave points', 'worst symmetry', 'worst fractal dimension',
'target']
```

and index =

```
RangeIndex(start=0, stop=569, step=1)
```

In [4]:

```
def answer_one():
    columns = ['mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean smoothness', 'mean compactnes
               'mean concavity', 'mean concave points', 'mean symmetry', 'mean fractal dimension', 'radius error'
               'texture error', 'perimeter error', 'area error', 'smoothness error', 'compactness error',
               'concavity error', 'concave points error', 'symmetry error', 'fractal dimension error', 'worst rad
               'worst texture', 'worst perimeter', 'worst area', 'worst smoothness', 'worst compactness',
               'worst concavity', 'worst concave points', 'worst symmetry', 'worst fractal dimension']
    df = pd.DataFrame(data=cancer.data, columns=columns)
    df['target'] = cancer.target
    return df

answer_one()
```

Out[4]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	wo are
0	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.300100	0.147100	0.2419	0.07871	...	17.33	184.60	201
1	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.086900	0.070170	0.1812	0.05667	...	23.41	158.80	195
2	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.197400	0.127900	0.2069	0.05999	...	25.53	152.50	170
3	11.420	20.38	77.58	386.1	0.14250	0.28390	0.241400	0.105200	0.2597	0.09744	...	26.50	98.87	567
4	20.290	14.34	135.10	1297.0	0.10030	0.13280	0.198000	0.104300	0.1809	0.05883	...	16.67	152.20	157
5	12.450	15.70	82.57	477.1	0.12780	0.17000	0.157800	0.080890	0.2087	0.07613	...	23.75	103.40	741
6	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.112700	0.074000	0.1794	0.05742	...	27.66	153.20	160
7	13.710	20.83	90.20	577.9	0.11890	0.16450	0.093660	0.059850	0.2196	0.07451	...	28.14	110.60	897
8	13.000	21.82	87.50	519.8	0.12730	0.19320	0.185900	0.093530	0.2350	0.07389	...	30.73	106.20	739
9	12.460	24.04	83.97	475.9	0.11860	0.23960	0.227300	0.085430	0.2030	0.08243	...	40.68	97.65	711
10	16.020	23.24	102.70	797.8	0.08206	0.06669	0.032990	0.033230	0.1528	0.05697	...	33.88	123.80	115
11	15.780	17.89	103.60	781.0	0.09710	0.12920	0.099540	0.066060	0.1842	0.06082	...	27.28	136.50	129
12	19.170	24.80	132.40	1123.0	0.09740	0.24580	0.206500	0.111800	0.2397	0.07800	...	29.94	151.70	133
13	15.850	23.95	103.70	782.7	0.08401	0.10020	0.099380	0.053640	0.1847	0.05338	...	27.66	112.00	876

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	wo are
14	13.730	22.61	93.60	578.3	0.11310	0.22930	0.212800	0.080250	0.2069	0.07682	...	32.01	108.80	69
15	14.540	27.54	96.73	658.8	0.11390	0.15950	0.163900	0.073640	0.2303	0.07077	...	37.13	124.10	94
16	14.680	20.13	94.74	684.5	0.09867	0.07200	0.073950	0.052590	0.1586	0.05922	...	30.88	123.40	11
17	16.130	20.68	108.10	798.8	0.11700	0.20220	0.172200	0.102800	0.2164	0.07356	...	31.48	136.80	13
18	19.810	22.15	130.00	1260.0	0.09831	0.10270	0.147900	0.094980	0.1582	0.05395	...	30.88	186.80	23
19	13.540	14.36	87.46	566.3	0.09779	0.08129	0.066640	0.047810	0.1885	0.05766	...	19.26	99.70	71
20	13.080	15.71	85.63	520.0	0.10750	0.12700	0.045680	0.031100	0.1967	0.06811	...	20.49	96.09	63
21	9.504	12.44	60.34	273.9	0.10240	0.06492	0.029560	0.020760	0.1815	0.06905	...	15.66	65.13	31
22	15.340	14.26	102.50	704.4	0.10730	0.21350	0.207700	0.097560	0.2521	0.07032	...	19.08	125.10	98
23	21.160	23.04	137.20	1404.0	0.09428	0.10220	0.109700	0.086320	0.1769	0.05278	...	35.59	188.00	26
24	16.650	21.38	110.00	904.6	0.11210	0.14570	0.152500	0.091700	0.1995	0.06330	...	31.56	177.00	22
25	17.140	16.40	116.00	912.7	0.11860	0.22760	0.222900	0.140100	0.3040	0.07413	...	21.40	152.40	14
26	14.580	21.53	97.41	644.8	0.10540	0.18680	0.142500	0.087830	0.2252	0.06924	...	33.21	122.40	89
27	18.610	20.25	122.10	1094.0	0.09440	0.10660	0.149000	0.077310	0.1697	0.05699	...	27.26	139.90	14
28	15.300	25.27	102.40	732.4	0.10820	0.16970	0.168300	0.087510	0.1926	0.06540	...	36.71	149.30	12
29	17.570	15.05	115.00	955.1	0.09847	0.11570	0.098750	0.079530	0.1739	0.06149	...	19.52	134.90	12
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
539	7.691	25.44	48.34	170.4	0.08668	0.11990	0.092520	0.013640	0.2037	0.07751	...	31.89	54.49	22
540	11.540	14.44	74.65	402.9	0.09984	0.11200	0.067370	0.025940	0.1818	0.06782	...	19.68	78.78	45
541	14.470	24.99	95.81	656.4	0.08837	0.12300	0.100900	0.038900	0.1872	0.06341	...	31.73	113.50	80
542	14.740	25.42	94.70	668.6	0.08275	0.07214	0.041050	0.030270	0.1840	0.05680	...	32.29	107.40	82
543	13.210	28.06	84.88	538.4	0.08671	0.06877	0.029870	0.032750	0.1628	0.05781	...	37.17	92.48	62

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	wo are
544	13.870	20.70	89.77	584.8	0.09578	0.10180	0.036880	0.023690	0.1620	0.06688	...	24.75	99.17	688
545	13.620	23.23	87.19	573.2	0.09246	0.06747	0.029740	0.024430	0.1664	0.05801	...	29.09	97.58	729
546	10.320	16.35	65.31	324.9	0.09434	0.04994	0.010120	0.005495	0.1885	0.06201	...	21.77	71.12	384
547	10.260	16.58	65.85	320.8	0.08877	0.08066	0.043580	0.024380	0.1669	0.06714	...	22.04	71.08	357
548	9.683	19.34	61.05	285.7	0.08491	0.05030	0.023370	0.009615	0.1580	0.06235	...	25.59	69.10	364
549	10.820	24.21	68.89	361.6	0.08192	0.06602	0.015480	0.008160	0.1976	0.06328	...	31.45	83.90	505
550	10.860	21.48	68.51	360.5	0.07431	0.04227	0.000000	0.000000	0.1661	0.05948	...	24.77	74.08	412
551	11.130	22.44	71.49	378.4	0.09566	0.08194	0.048240	0.022570	0.2030	0.06552	...	28.26	77.80	436
552	12.770	29.43	81.35	507.9	0.08276	0.04234	0.019970	0.014990	0.1539	0.05637	...	36.00	88.10	594
553	9.333	21.94	59.01	264.0	0.09240	0.05605	0.039960	0.012820	0.1692	0.06576	...	25.05	62.86	295
554	12.880	28.92	82.50	514.3	0.08123	0.05824	0.061950	0.023430	0.1566	0.05708	...	35.74	88.84	595
555	10.290	27.61	65.67	321.4	0.09030	0.07658	0.059990	0.027380	0.1593	0.06127	...	34.91	69.57	357
556	10.160	19.59	64.73	311.7	0.10030	0.07504	0.005025	0.011160	0.1791	0.06331	...	22.88	67.88	347
557	9.423	27.88	59.26	271.3	0.08123	0.04971	0.000000	0.000000	0.1742	0.06059	...	34.24	66.50	330
558	14.590	22.68	96.39	657.1	0.08473	0.13300	0.102900	0.037360	0.1454	0.06147	...	27.27	105.90	733
559	11.510	23.93	74.52	403.5	0.09261	0.10210	0.111200	0.041050	0.1388	0.06570	...	37.16	82.28	474
560	14.050	27.15	91.38	600.4	0.09929	0.11260	0.044620	0.043040	0.1537	0.06171	...	33.17	100.20	706
561	11.200	29.37	70.67	386.0	0.07449	0.03558	0.000000	0.000000	0.1060	0.05502	...	38.30	75.19	439
562	15.220	30.62	103.40	716.9	0.10480	0.20870	0.255000	0.094290	0.2128	0.07152	...	42.79	128.70	915
563	20.920	25.09	143.00	1347.0	0.10990	0.22360	0.317400	0.147400	0.2149	0.06879	...	29.41	179.10	181
564	21.560	22.39	142.00	1479.0	0.11100	0.11590	0.243900	0.138900	0.1726	0.05623	...	26.40	166.10	202
565	20.130	28.25	131.20	1261.0	0.09780	0.10340	0.144000	0.097910	0.1752	0.05533	...	38.25	155.00	173

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	wo are
566	16.600	28.08	108.30	858.1	0.08455	0.10230	0.092510	0.053020	0.1590	0.05648	...	34.12	126.70	112
567	20.600	29.33	140.10	1265.0	0.11780	0.27700	0.351400	0.152000	0.2397	0.07016	...	39.42	184.60	182
568	7.760	24.54	47.92	181.0	0.05263	0.04362	0.000000	0.000000	0.1587	0.05884	...	30.37	59.16	268

569 rows × 31 columns

Question 2

What is the class distribution? (i.e. how many instances of `malignant` (encoded 0) and how many `benign` (encoded 1)?)

This function should return a Series named `target` of length 2 with integer values and index = `['malignant', 'benign']`

In [5]:
```
def answer_two():
    cancerdf = answer_one()
    count = cancerdf['target'].groupby(cancerdf['target']).count()
    return pd.Series(count.values, index=['malignant', 'benign'])

answer_two()
```

Out[5]:
```
malignant    212
benign       357
dtype: int64
```

Question 3

Split the DataFrame into `X` (the data) and `y` (the labels).

This function should return a tuple of length 2: `(X, y)`*, where*

- `X` *has shape* `(569, 30)`
- `y` *has shape* `(569,)`.

In [6]:
```
def answer_three():
    cancerdf = answer_one()
    X = cancerdf.iloc[:, :-1]
    y = cancerdf.iloc[:, -1]
    return X, y

answer_three()
```

Out[6]:
```
(     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
 0         17.990         10.38          122.80     1001.0          0.11840
 1         20.570         17.77          132.90     1326.0          0.08474
 2         19.690         21.25          130.00     1203.0          0.10960
 3         11.420         20.38           77.58      386.1          0.14250
 4         20.290         14.34          135.10     1297.0          0.10030
 5         12.450         15.70           82.57      477.1          0.12780
 6         18.250         19.98          119.60     1040.0          0.09463
 7         13.710         20.83           90.20      577.9          0.11890
 8         13.000         21.82           87.50      519.8          0.12730
 9         12.460         24.04           83.97      475.9          0.11860
 10        16.020         23.24          102.70      797.8          0.08206
 11        15.780         17.89          103.60      781.0          0.09710
 12        19.170         24.80          132.40     1123.0          0.09740
 13        15.850         23.95          103.70      782.7          0.08401
 14        13.730         22.61           93.60      578.3          0.11310
 15        14.540         27.54           96.73      658.8          0.11390
 16        14.680         20.13           94.74      684.5          0.09867
 17        16.130         20.68          108.10      798.8          0.11700
```

Question 4

Using `train_test_split`, split `X` and `y` into training and test sets `(X_train, X_test, y_train, and y_test)`.

Set the random number generator state to 0 using `random_state=0` to make sure your results match the autograder!

This function should return a tuple of length 4: `(X_train, X_test, y_train, y_test)`*, where*

- `X_train` *has shape* `(426, 30)`
- `X_test` *has shape* `(143, 30)`
- `y_train` *has shape* `(426,)`

- `y_test` *has shape* `(143,)`

In [7]:

```python
from sklearn.model_selection import train_test_split

def answer_four():
    X, y = answer_three()
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
    return X_train, X_test, y_train, y_test

answer_four()
```

441	17.270	25.42	112.40	928.8	0.08331
137	11.430	15.39	73.06	399.8	0.09639
230	17.050	19.08	113.40	895.0	0.11410
7	13.710	20.83	90.20	577.9	0.11890
408	17.990	20.66	117.80	991.7	0.10360
523	13.710	18.68	88.73	571.0	0.09916
361	13.300	21.57	85.24	546.1	0.08582
553	9.333	21.94	59.01	264.0	0.09240
478	11.490	14.59	73.99	404.9	0.10460
303	10.490	18.61	66.86	334.3	0.10680
..	...	...	...	...	...
459	9.755	28.20	61.68	290.9	0.07984
510	11.740	14.69	76.31	426.0	0.08099
151	8.219	20.70	53.27	203.9	0.09405
244	19.400	23.50	129.10	1155.0	0.10270
543	13.210	28.06	84.88	538.4	0.08671
544	13.870	20.70	89.77	584.8	0.09578
265	20.730	31.12	135.70	1419.0	0.09469

Question 5

Using KNeighborsClassifier, fit a k-nearest neighbors (knn) classifier with `X_train`, `y_train` and using one nearest neighbor (`n_neighbors = 1`).

This function should return a `sklearn.neighbors.classification.KNeighborsClassifier`.

In [14]:
```
from sklearn.neighbors import KNeighborsClassifier

def answer_five():
    X_train, X_test, y_train, y_test = answer_four()
    knn = KNeighborsClassifier(n_neighbors = 1)
    return knn.fit(X_train, y_train)

answer_five()
```

Out[14]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Question 6

Using your knn classifier, predict the class label using the mean value for each feature.

Hint: You can use `cancerdf.mean()[:-1].values.reshape(1, -1)` which gets the mean value for each feature, ignores the target column, and reshapes the data from 1 dimension to 2 (necessary for the precict method of KNeighborsClassifier).

This function should return a numpy array either `array([ 0.])` or `array([ 1.])`

In [18]:
```
cancerdf = answer_one()
means = cancerdf.mean()[:-1].values.reshape(1, -1)
means
```

Out[18]:
```
array([[  1.41272917e+01,   1.92896485e+01,   9.19690334e+01,
          6.54889104e+02,   9.63602812e-02,   1.04340984e-01,
          8.87993158e-02,   4.89191459e-02,   1.81161863e-01,
          6.27976098e-02,   4.05172056e-01,   1.21685343e+00,
          2.86605923e+00,   4.03370791e+01,   7.04097891e-03,
          2.54781388e-02,   3.18937163e-02,   1.17961371e-02,
          2.05422988e-02,   3.79490387e-03,   1.62691898e+01,
          2.56772232e+01,   1.07261213e+02,   8.80583128e+02,
          1.32368594e-01,   2.54265044e-01,   2.72188483e-01,
          1.14606223e-01,   2.90075571e-01,   8.39458172e-02]])
```

In [21]:
```
def answer_six():
    cancerdf = answer_one()
    means = cancerdf.mean()[:-1].values.reshape(1, -1)
    X_train, X_test, y_train, y_test = answer_four()
    knn = KNeighborsClassifier(n_neighbors = 1)
    knn.fit(X_train, y_train)
    prediction = knn.predict(means)
    return prediction

answer_six()
```

Out[21]:
```
array([1])
```

Question 7

Using your knn classifier, predict the class labels for the test set `X_test`.

This function should return a numpy array with shape `(143,)` and values either `0.0` or `1.0`.

In [23]:
```
def answer_seven():
    X_train, X_test, y_train, y_test = answer_four()
    knn = answer_five()
    prediction = knn.predict(X_test)
    return prediction

answer_seven()
```

Out[23]:
```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 0])
```

Question 8

Find the score (mean accuracy) of your knn classifier using `X_test` and `y_test`.

This function should return a float between 0 and 1

In [24]:
```
def answer_eight():
    X_train, X_test, y_train, y_test = answer_four()
    knn = answer_five()
    return knn.score(X_test, y_test)

answer_eight()
```

Out[24]: 0.91608391608391604

Optional plot

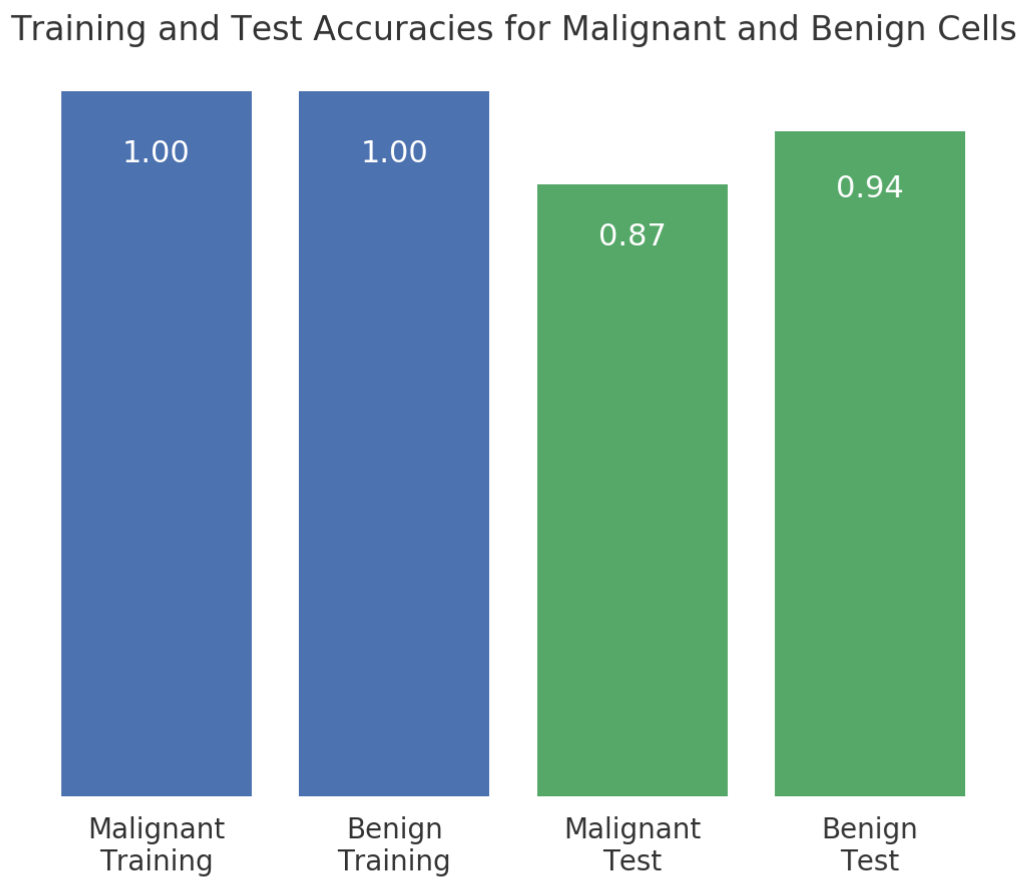
Try using the plotting function below to visualize the differet predicition scores between training and test sets, as well as malignant and benign cells.

In [12]:
```
def accuracy_plot():
    import matplotlib.pyplot as plt

    %matplotlib notebook

    X_train, X_test, y_train, y_test = answer_four()

    # Find the training and testing accuracies by target value (i.e. malignant, benign)
    mal_train_X = X_train[y_train==0]
    mal_train_y = y_train[y_train==0]
    ben_train_X = X_train[y_train==1]
    ben_train_y = y_train[y_train==1]

    mal_test_X = X_test[y_test==0]
    mal_test_y = y_test[y_test==0]
    ben_test_X = X_test[y_test==1]
    ben_test_y = y_test[y_test==1]

    knn = answer_five()

    scores = [knn.score(mal_train_X, mal_train_y), knn.score(ben_train_X, ben_train_y),
              knn.score(mal_test_X, mal_test_y), knn.score(ben_test_X, ben_test_y)]


    plt.figure()

    # Plot the scores as a bar chart
    bars = plt.bar(np.arange(4), scores, color=['#4c72b0','#4c72b0','#55a868','#55a868'])

    # directly label the score onto the bars
    for bar in bars:
        height = bar.get_height()
        plt.gca().text(bar.get_x() + bar.get_width()/2, height*.90, '{0:.{1}f}'.format(height, 2),
                     ha='center', color='w', fontsize=11)

    # remove all the ticks (both axes), and tick labels on the Y axis
    plt.tick_params(top='off', bottom='off', left='off', right='off', labelleft='off', labelbottom='on')

    # remove the frame of the chart
    for spine in plt.gca().spines.values():
        spine.set_visible(False)
```

```
    plt.xticks([0,1,2,3], ['Malignant\nTraining', 'Benign\nTraining', 'Malignant\nTest', 'Benign\nTest'], alpha=
    plt.title('Training and Test Accuracies for Malignant and Benign Cells', alpha=0.8)
```

In [25]:
```
# Uncomment the plotting function to see the visualization,
# Comment out the plotting function when submitting your notebook for grading

accuracy_plot()
```

Figure 1



In [ ]:

```
```